**Lab: Auditing with Lighthouse**

<https://developers.google.com/web/ilt/pwa/lab-auditing-with-lighthouse>

**Overview**

This lab shows you how you can use [Lighthouse](https://developers.google.com/web/tools/lighthouse), an [open-source](https://github.com/GoogleChrome/lighthouse) tool from Google, to audit a web app for PWA features. Lighthouse provides a set of metrics to help guide you in building a PWA with a full application-like experience for your users.

**What you will learn**

* How to use Lighthouse to audit your progressive web apps

**What you should know**

* Basic JavaScript and HTML

**What you need before you begin**

* Connection to the internet
* [Chrome](https://www.google.com/chrome/browser/desktop/) 52+ browser
* Node v6+
* A text editor

**1. Get set up**

If you have not downloaded the repository and installed the [LTS version of Node.js](https://nodejs.org/en/), follow the instructions in [Setting up the labs](https://developers.google.com/web/ilt/pwa/setting-up-the-labs).

If you don't have a preferred local development server, install the Node.js http-server package:

npm install http-server -g

Navigate into the lighthouse-lab/app/ directory and start the server:

cd lighthouse-lab/app  
http-server -p 8080 -a localhost -c 0

You can terminate the server at any time with Ctrl-c.

Open your browser and navigate to localhost:8080/.

**Note:** [Unregister](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#unregister) any service workers and [clear all service worker caches](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#clearcache) for localhost so that they do not interfere with the lab.

If you have a text editor that lets you open a project, open the lighthouse-lab/app/ folder. This will make it easier to stay organized. Otherwise, open the folder in your computer's file system. The app/ folder is where you will be building the lab.

This folder contains:

* css/main.css is the cascading stylesheet for the application
* images/ folder contains images for the app and home screen icon
* index.html is the main HTML page for our sample site/application

**2. Install Lighthouse**

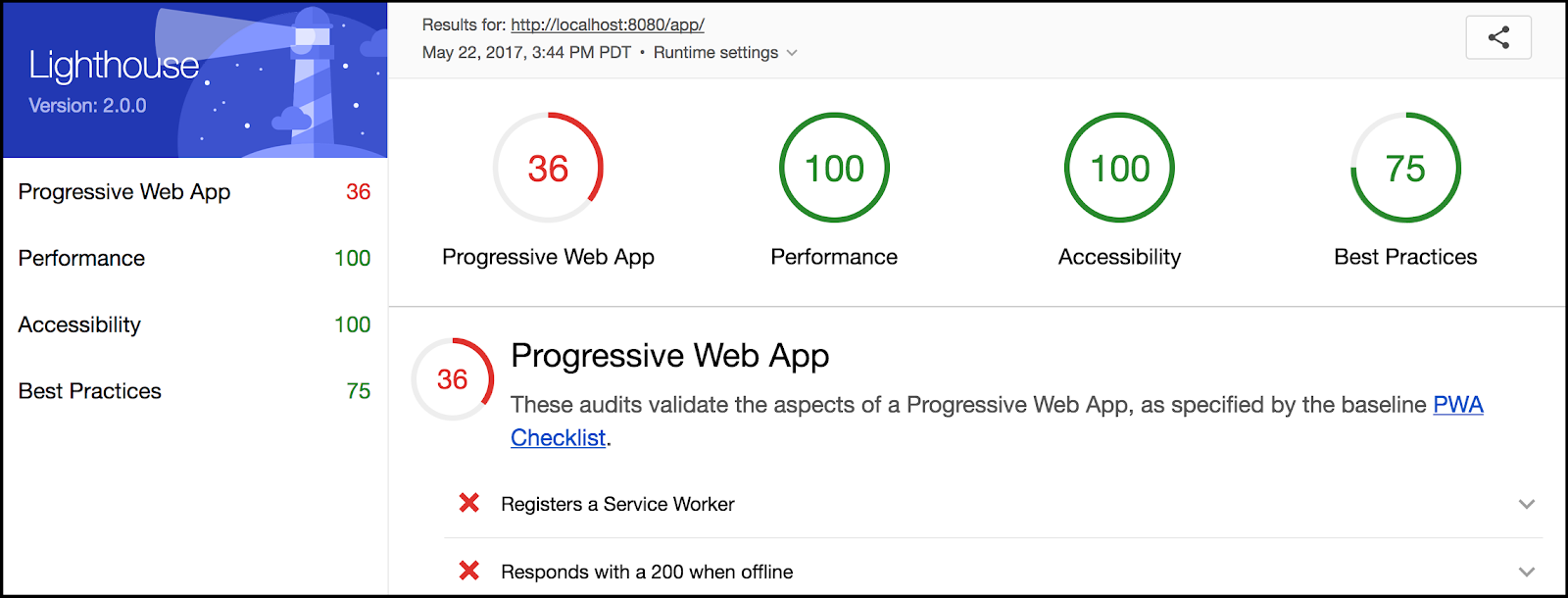
Lighthouse is available as a Chrome extension for [Chrome](https://www.google.com/chrome/browser/desktop/index.html) 52 and later.

Download the Lighthouse Chrome extension from the [Chrome Web Store](http://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk). When installed it places an  icon in your taskbar.

**3. Test the app**

In the browser (on the app/ page) click the Lighthouse icon and choose **Generate report** (you may be prompted to close Developer Tools if they are open).

Lighthouse runs the report and generates an HTML page with the results. The report page should look similar to this:



**Note:** The UI for Lighthouse is still being updated, so your report may not look exactly like this one.

Looks like we have a pretty low score (your score may not match exactly). Take a moment to look through the report and see what is missing.

**4. Adding a manifest file**

The report indicates that we need a manifest file.

**4.1 Create the manifest file**

Create an empty file called manifest.json in the app/ directory.

Replace TODO 4.1 in index.html with the following:

<!-- Web Application Manifest -->  
<link rel="manifest" href="manifest.json">

**4.2 Add manifest code**

Add the following to the manifest.json file:

{  
  "name": "Demo Blog Application",  
  "short\_name": "Blog",  
  "start\_url": "index.html",  
  "icons": [{  
        "src": "images/touch/icon-128x128.png",  
        "sizes": "128x128",  
        "type": "image/png"  
      }, {  
        "src": "images/touch/icon-192x192.png",  
        "sizes": "192x192",  
        "type": "image/png"  
      }, {  
        "src": "images/touch/icon-256x256.png",  
        "sizes": "256x256",  
        "type": "image/png"  
      }, {  
        "src": "images/touch/icon-384x384.png",  
        "sizes": "384x384",  
        "type": "image/png"  
      }, {  
        "src": "images/touch/icon-512x512.png",  
        "sizes": "512x512",  
        "type": "image/png"  
      }],  
  "background\_color": "#3E4EB8",  
  "display": "standalone",  
  "theme\_color": "#2E3AA1"  
}

**4.3 Add tags for other browsers**

Replace TODO 4.3 in index.html with the following:

<!-- Chrome for Android theme color -->  
<meta name="theme-color" content="#2E3AA1">  
  
<!-- Add to homescreen for Chrome on Android -->  
<meta name="mobile-web-app-capable" content="yes">  
<meta name="application-name" content="Blog">  
<link rel="icon" sizes="192x192" href="images/touch/icon-192x192.png">  
  
<!-- Add to homescreen for Safari on iOS -->  
<meta name="apple-mobile-web-app-capable" content="yes">  
<meta name="apple-mobile-web-app-status-bar-style" content="black">  
<meta name="apple-mobile-web-app-title" content="Blog">  
<link rel="apple-touch-icon" href="images/touch/icon-192x192.png">  
  
<!-- Tile for Win8 -->  
<meta name="msapplication-TileColor" content="#3372DF">  
<meta name="msapplication-TileImage" content="images/touch/icon-192x192.png">

**Explanation**

We have created a manifest file and "add to homescreen" tags. Don't worry about the details of the manifest and these tags. Here is how they work:

1. Chrome uses manifest.json to know how to style and format some of the progressive parts of your app, such as the "add to homescreen" icon and splash screen.
2. Other browsers don't (currently) use the manifest.json file to do this, and instead rely on HTML tags for this information. While Lighthouse doesn't require these tags, we've added them because they are important for supporting as many browsers as possible.

This lets us satisfy the manifest related requirements of Lighthouse (and a PWA).

**For more information**

* [Add to home screen](https://developer.chrome.com/multidevice/android/installtohomescreen)
* [Web app manifests](https://developers.google.com/web/updates/2014/11/Support-for-installable-web-apps-with-webapp-manifest-in-chrome-38-for-Android)

**5. Adding a service worker**

We can see from the report that having a service worker is necessary.

**5.1 Register a service worker**

Create an empty JavaScript file in the root directory (app/) and name it service-worker.js. This is going to be our service worker file.

Now replace TODO 5.1 in **index.html** with the following and save the file:

<script>  
  (function() {  
    if (!('serviceWorker' in navigator)) {  
      console.log('Service worker not supported');  
      return;  
    }  
    navigator.serviceWorker.register('service-worker.js')  
    .then(function(registration) {  
      console.log('SW successfully registered');  
    })  
    .catch(function(error) {  
      console.log('registration failed', error);  
    });  
  })();  
</script>

**5.2 Caching offline & start pages**

The report also indicates that our app must respond with a 200 when offline and must have our starting URL ("start\_url") cached.

Add the following code to the empty service-worker.js file (which should be at app/service-worker.js):

self.addEventListener('install', function(event) {  
  event.waitUntil(  
    caches.open('static-cache-v1')  
    .then(function(cache) {  
      return cache.addAll([  
        '.',  
        'index.html',  
        'css/main.css',  
        'https://fonts.googleapis.com/css?family=Roboto:300,400,500,700',  
        'images/still\_life-1600\_large\_2x.jpg',  
        'images/still\_life-800\_large\_1x.jpg',  
        'images/still\_life\_medium.jpg',  
        'images/still\_life\_small.jpg'  
      ]);  
    })  
  );  
});  
  
self.addEventListener('fetch', function(event) {  
  event.respondWith(caches.match(event.request)  
  .then(function(response) {  
      if (response) {  
        return response;  
      }  
      return fetch(event.request);  
    })  
  );  
});

Save the file and refresh the page (for the app, not the Lighthouse page). Check the console and confirm that the service worker has registered successfully.

**Explanation**

We have created a service worker for our app and registered it. Here is what it does:

1. The first block (install event listener) caches the files our app's files, so that they are saved locally. This lets us access them even when offline, which is what the next block does.
2. The second block (fetch event listener) intercepts requests for resources and checks first if they are cached locally. If they are, the browser gets them from the cache without needing to make a network request. This lets us respond with a 200 even when offline.

Once we have loaded the app initially, all the files needed to run the app are saved in the cache. If the page is loaded again, the browser grabs the files from the cache regardless of network conditions. This also lets us satisfy the requirement of having our starting URL (index.html) cached.

**Solution code**

To get a copy of the working code, navigate to the solution/ folder.

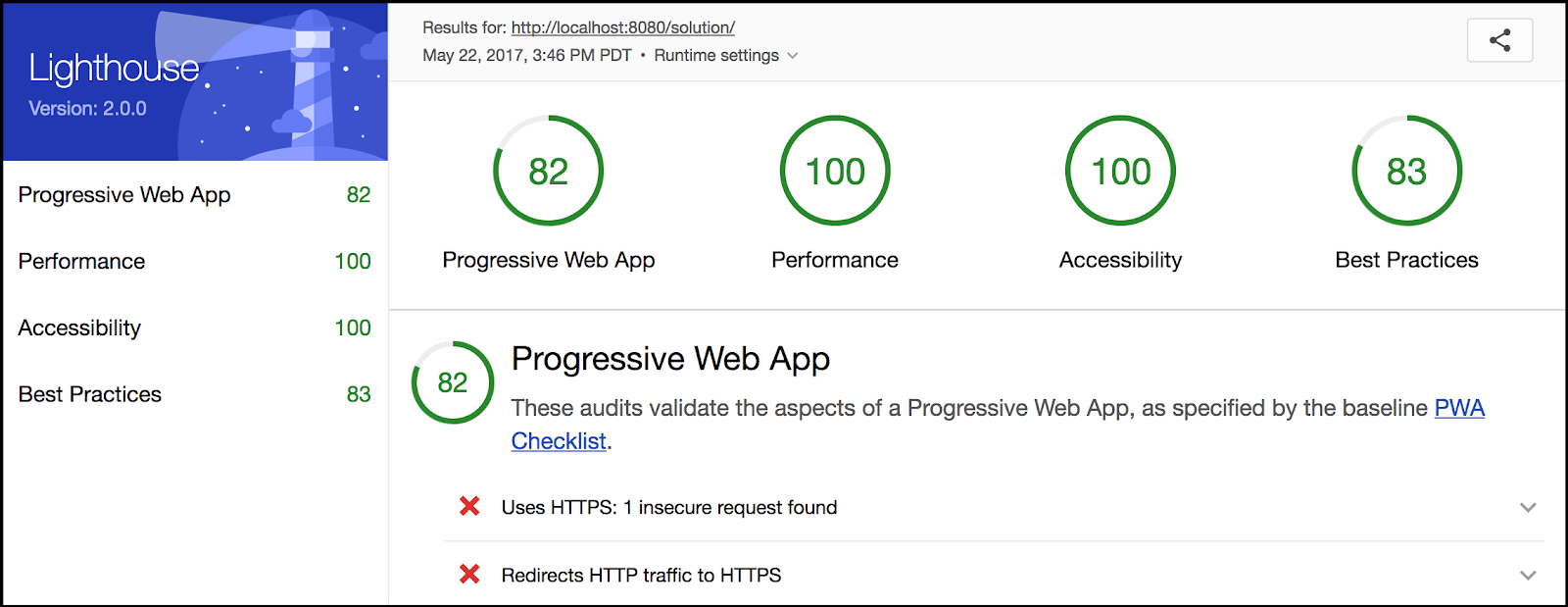
**6. Test the updated app**

Now we need to retest the app to see our changes. Return to the localhost page where that app is hosted. Click the Lighthouse icon and generate a report (you may be prompted to close Developer Tools if they are still open).

Now we should have passed many more tests.

**Note:** You may need to [disable the browser cache](https://developers.google.com/web/ilt/pwa/tools-for-pwa-developers#disablehttpcache) to see the improved results. Then refresh the app and run Lighthouse again.

The report should look something like this:



Now our score is much better (your score may not match exactly).

You can see that we are still missing the HTTPS requirements, since we are using a local server. In production, service workers require HTTPS, so you'll need to use that.

**7. Optional: Run Lighthouse from the command line**

If you want to run Lighthouse from the command line (for example, to integrate it with a build process) it is available as a Node module that you can install from the terminal.

If you haven't already, [download Node](https://nodejs.org/en/) and select the Long Term Support (LTS) version that best suits your environment and operating system (Lighthouse requires Node v6 or greater).

Install Lighthouse's Node module from the command line:

npm install -g lighthouse

Run Lighthouse on a demo progressive web app:

lighthouse https://airhorner.com/

Or on the app that you just made (note that your localhost port may be different):

lighthouse http://localhost:8080/lighthouse-lab/app/

You can check Lighthouse flags and options with:

lighthouse --help

The lighthouse command line tool will generate an HTML (the same as the Chrome extension) in the working directory. You can then open the file with your browser.

**Congratulations!**

You have learned how to use the Lighthouse tool to audit your progressive web apps.